1. Solution description

a. Backbone of the solution

First issue of the implementation was to choose the best MRCNN implementation. We chose MRCNN algorithm in the first part of the project as the most effective, according to available publications.
Parameters which we considered were:

- number of stars on github

GitHub Stars are an easy metric to keep track of, and We've used them to measure how popular an open source project is. Often using Stars is good to determine how battle-tested a repo is, under the assumption that a repo that has received more Stars must have received more use.

- number of users in other repositories
- number of forks

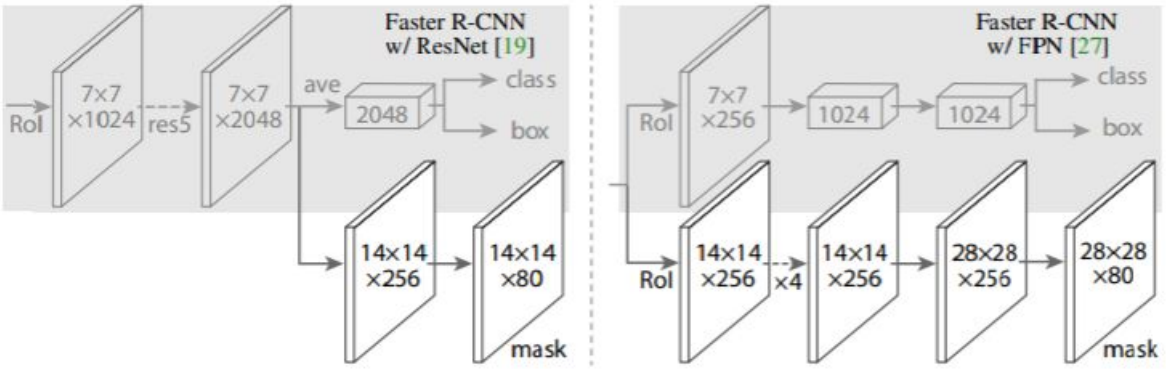
These parameters let us choose the most popular library, which probably would be the easiest to use, with most of bug fixed and with good community around.
Considering also development background of the team we decided to choose implementation in Tensorflow.
Another important aspect was a well written documentation.
Parameters of these requirements are as follows:

Repository	Stars	Users	Forks	Technology
matterport/Mask_RCNN	15500	47	7200	Tensorflow
Facebook maskrcnn	7200	-	2100	PyTorch
FasterRCNN	5200	307	1600	Tensorflow

According to the following parameters and choosing between available libraries we chose matterport's repository. It is the most often used and use a technology which we know.
The implementation was based on Mask RCNN publication.

b. Detailed architecture of the neural network

The architecture of the chosen implementation is as follows:



It extends two existing Faster RCNN heads. Left/Right panels show the heads for the ResNet C4 and FPN backbones. Numbers denote spatial resolution and channels. Arrows denote either conv, deconv, or fc layers as can be inferred from context (conv preserves spatial dimension while deconv increases it). All convs are 3×3, except the output conv which is 1×1, deconvs are 2×2 with stride 2, and there is used ReLU in hidden layers.
Left: 'res5' denotes ResNet's fifth stage, which for simplicity were altered so that the first conv operates on a 7×7 RoI with stride 1 (instead of 14×14 / stride 2 as in [19]).
Right: '×4' denotes a stack of four consecutive convs.

c. Tools and libraries

Project was developed with python in version 3.6.8.
In order to use this repository the user musts follow some requirements about the software. We used in the project following libraries:

- numpy
- scipy
- Pillow
- cython
- matplotlib
- scikit-image
- tensorflow==1.3.0
- keras==2.0.8
- opencv-python
- h5py
- imgaug
- tensorboard

Implementation of the MRCNN lets you use tensorflow and keras in newer version but we do not recommend it.

2. Installation and usage

a. Using our neural network

1.In PythonAPI directory install dependencies:
pip3 install -r requirements.txt

2.Paste photos which you want to validate to the *real_test* directory

3.Run human_recognition.py script

b. Initializing your own neural network:

1. Download datasets and annotations from:
http://cocodataset.org/#download

2. Place annotations to annotation directory

3. In PythonAPI directory install dependencies:
pip3 install -r requirements.txt

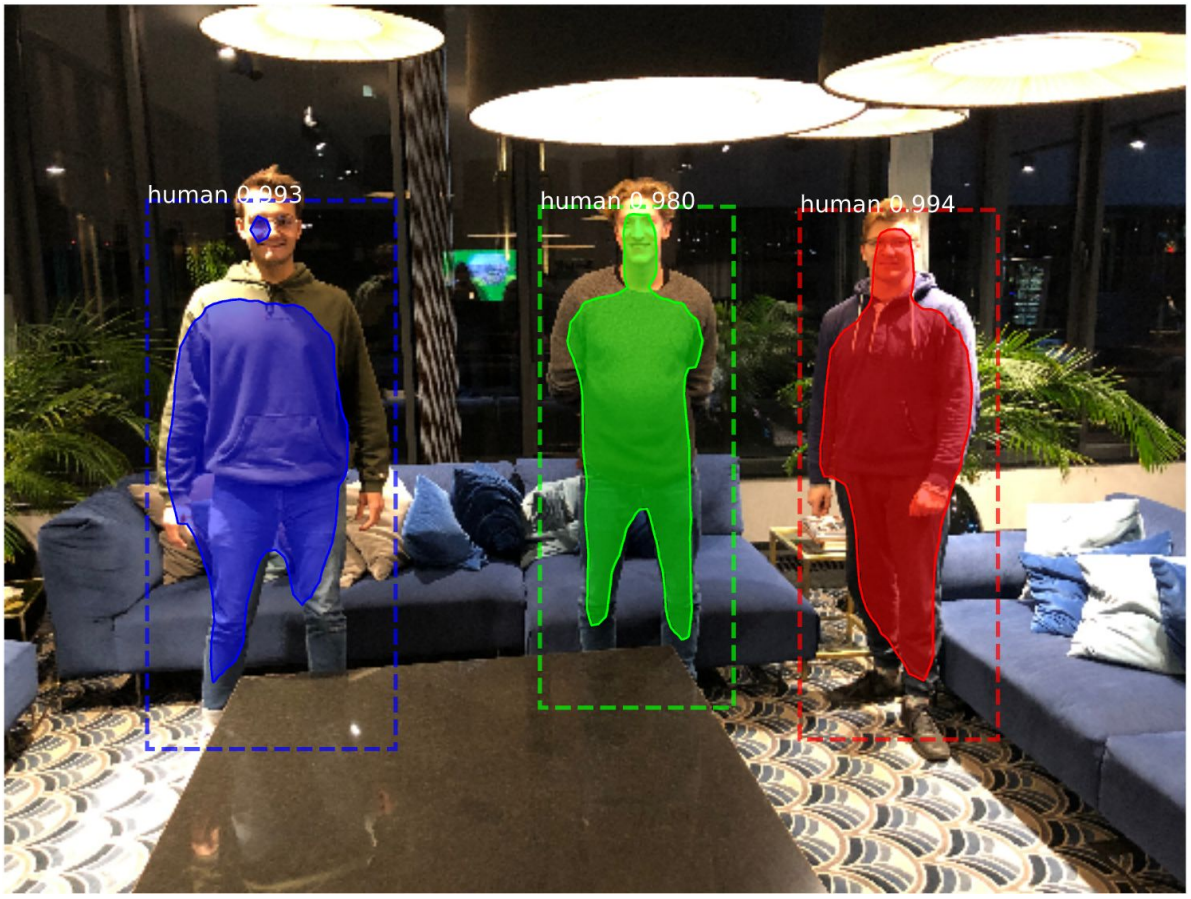
4. Paste photos which you want to validate to the *real_test* directory

5. Run human_recognition.py script with arguments:
- type of annotations for validation dataset
- type of annotations for test dataset
- directory of training photos directory
- directory of validation photos directory
- directory of photos for real test
- initial weights for network

So example invocation would look like:
python3 new_recognition_script.py ‘val2017’ ‘train2017’ '/media/dawid/MAJKI/train2017' '/home/dawid/Desktop/human_images' '../real_test' '../initial_weights/initial_weights.h5'

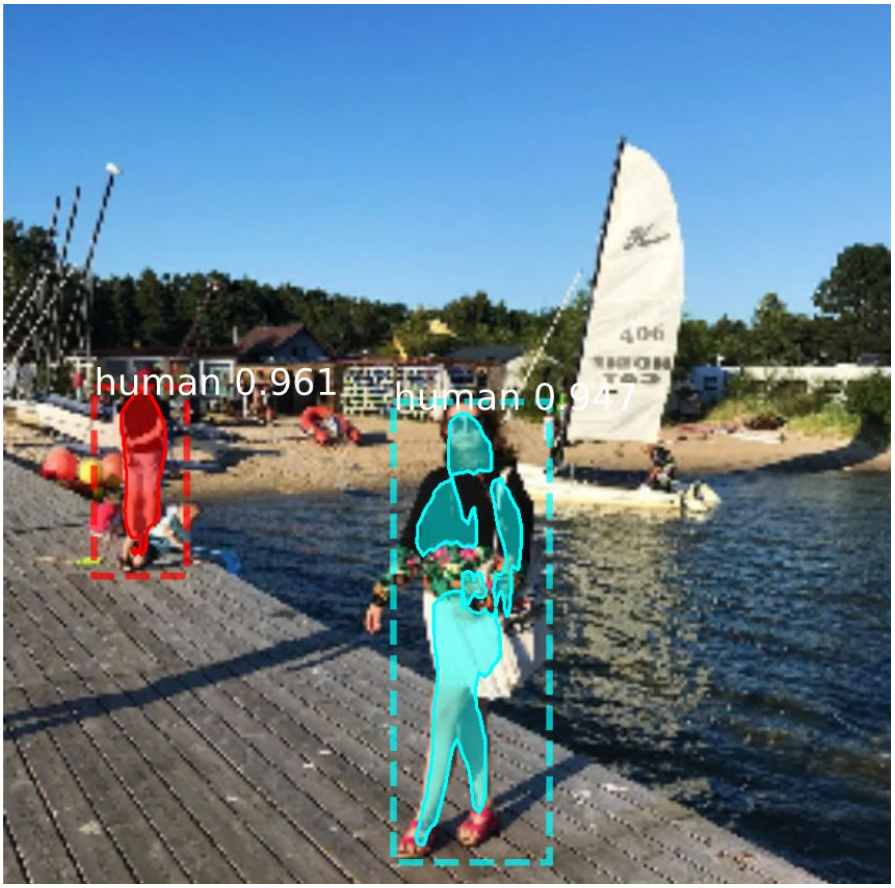
3. Results

After training model ability to detect human instances on pictures was evaluated. Couple pictures prepared by authors were processed with network to prove training success. As a result of detection our model returns mask in the form of 2-dimensional array with boolean values. “True” values of formed mask correspond to pixels on input images associated with human instance. Bounding box is formed basing on the resulting mask.



Proud authors of human detection model

Boy in skiing gear



Postcard from the seaside

Shibuya crossing, Tokyo, Japan

Surfer catching a perfect wave

Football match

Workers

4. Efficiency of the model measured with self-implemented IoU method

After visual demonstration of implemented classifier it is time to measure its efficiency. Included library didn't provide any suitable testing method, so it was necessary to write our own method that measures pixel accuracy by comparing resultant mask with one defined in the validation dataset.
We used i**ntersection over union method** that allows to measure approximate percentage of mask accuracy. It's principle of operation is to compute ratio between common part and joint of both masks. Pictures below present mentioned operations.

the IoU metric measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across *both* masks.

$$IoU = \frac{target \cap prediction}{target \cup prediction}$$

Mask from dataset

Output mask

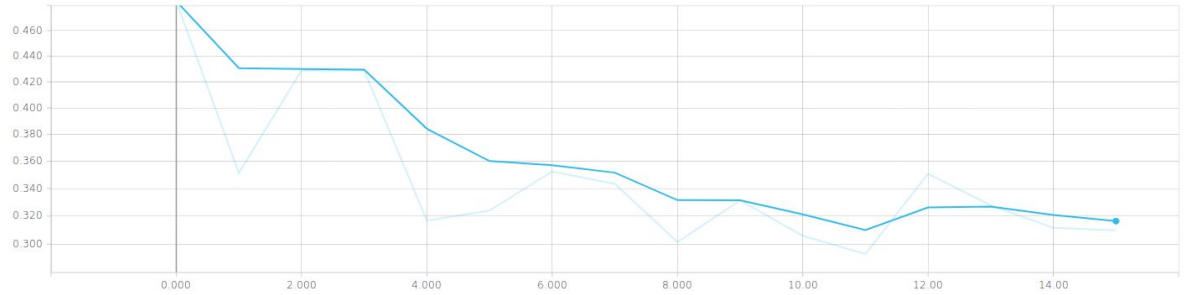
Common part (intersection) Joint (union)

More detailed description can be found under this address: [Evaluating image segmentation models](#) .

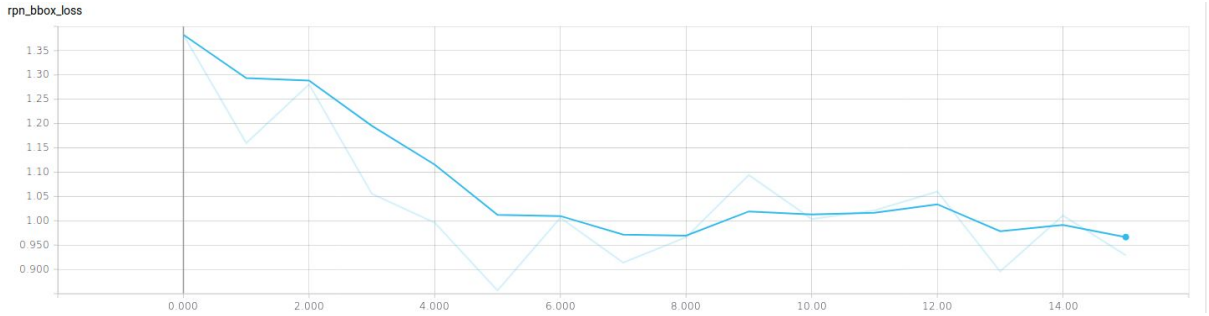
In next step we count pixels belonging to intersection mask and union mask. Division of intersection over union, multiplied by 100 gives us percentage measure of output accuracy. Metric returns 100% if detection result is identical with example defined in dataset and 0% if result is not contained in a desired mask.
It should be noted that all non 0% results are satisfying. Even residual masks allows us to create bounding box of instance on image, which is recognized as valid human detection. Test was performed on over 1000 testing images from COCO dataset. Mean of intersection over union metric results was 29,5%. Only 82 classifications were rated 0,0%, which gives us over 90% efficiency in human detection task.

5. Progress of the neural network in the training on training and validation data

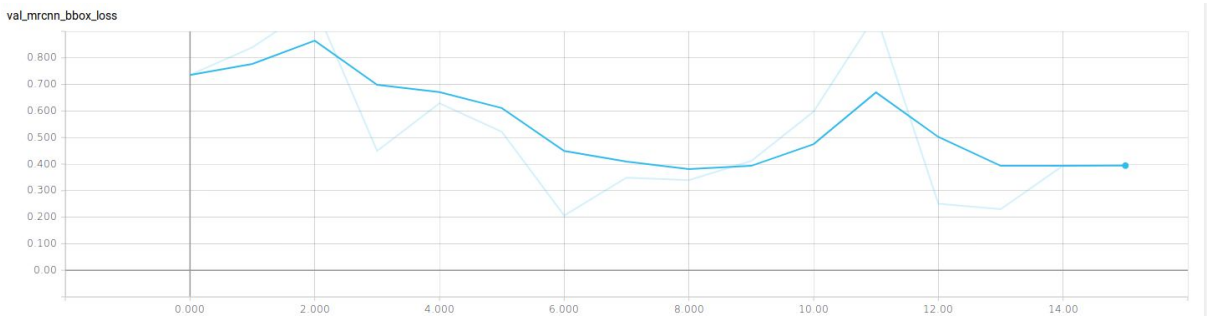
Additionally during the training process we measured a progress of the neural network online.
Logged data was gathered with Tensorboard tool. It is very popular tool. Additionally it is compatible with tensorflow and keras libraries. The results are as follows:



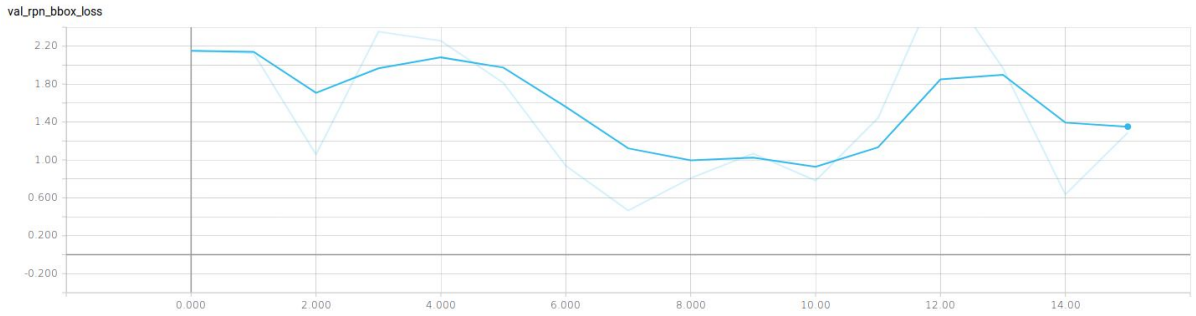
Loss function of the network Mask Faster R-CNN on training data



Loss function of the network Region Proposal Network (RPN) — Backbone of Mask Faster R-CNN on training data



Loss function of the network Mask Faster R-CNN on validation data



Loss function of the network Region Proposal Network (RPN) — Backbone of Mask Faster R-CNN on validation data

Above diagrams let use see the progress of the model. It also shows the overfitting of the model. It is easy to notice that model after 10 epoch starts to overfit to training data. Loss function on validation data is growing. It is clear signal that we should stop training process here and we should choose model with the lowest loss fucntion value until this time. It was done. Sample results are based on this, the best model.

6. Additional thoughts

Used repository don't implement proper way to evaluate new dataset. We had trouble using it on chosen dataset. As we read in issue forum we are not alone with it. That's a reason why we wrote our own method to examine it on other datasets.

Mask RCNN in very effective way of classifying people. Used algorithm can be easly used to recognize other objects like trucks, signs ect. The only limitation is availability and quality of the dataset.

The process of training neural network is time-consuming. It takes a few hours depending of the number of epochs and batch size. Trening on the GPU is faster.